International Journal of Applied Mathematics
& Statistical Sciences (IJAMSS)
ISSN(P): 2319-3972; ISSN(E): 2319-3980
Vol. 5, Issue 4, Jun - Jul 2016; 93-112

International Academy of Science, Engineering and Technology
Connecting Researchers; Nurturing Innovations

TEXT MINING OF EMAILS USING MATLAB CODES

VADIRAJA, N

Assistant Professor – Statistics, Mysore Medical College and Research Institute, Mysuru, Karnataka, India

ABSTRACT

As an application to text mining, it is developed that a Statistical SPAM Filter using Matlab codes. SPAM is defined as 'unwanted Emails'. This is very useful when a workstation receives more number of emails including SPAMs into the email box. Question is how to get rid off from SPAMs. More the SPAMs, greater is the chance of ignoring HAMs (Legitimate Emails). SPAM filters classify emails into SPAMs or HAMs. However, misclassifying hams is worse than misclassifying SPAMs. As a result, it is interested to get rid off from SPAMs keeping in mind that no legitimate emails are ignored. Existing, now-a-days, lots of rule based SPAM filters, however our interest is to develop statistical filter that uses Bayesian and SBPH approach on emails and to compare. Matlab codes are used to generate the test programme.

KEYWORDS: Statistical SPAM Filter Using Matlab Codes

INTRODUCTION

SPAM is defined as 'unwanted Emails'. The problem arises when a person receives more number of SPAMs. More the SPAMs, greater is the chance of HAMs being unnoticed. So, the question is how to get rid off from SPAMs. On the other hand it is well known that spamming is one of the effective ways of email marketing due to its lesser cost. As a result and high-reach endeavor , for very little investment and effort, spammers can reach millions of potential customers via bunch emails, and just 100 responses out of 10 million can turn them a profit. Recently, the Pew Internet & American Life Project reported that seven percent of Americans have purchased something through unsolicited emails. Thus, companies have started using this method for effective selling of their products. Some companies, in order to maintain their prestige, started using third parties. Whatever may be the reason, the idea behind SPAMs is sales. As a result most of the SPAMs are sales pitched.

On the other hand, same companies started using Spam filters to get rid off from the problem of receiving many SPAMs and to increase their productivity time and profit by not spending there time in unnoticing hams, in finding and deleting SPAMs etc. Based on the above two points, one wants to filter SPAMs (Non-spammer) and another (Spammer) wants to find a way to beat filters. Thus, a stage is set for the war between spammers and Anti-spammers. Statistics says that, according to Feb, 2007 survey, spamming increased to 90 billion per day, almost a hike of 60 billion per day during 2005 [8].

At the receiver's end, non-spammers have following problems.

Missing innocent mails is worse than receiving SPAMS.

More SPAMs a user gets, the less likely to notice innocent mails.

Better the SPAM filter, more likely the user ignore everything they catch.

In order to overcome the above difficulties, in recent times anti-spammers have come out with a rule based and statistical filters. As a result, Spammers followed the following methods to beat the filters to percolate the message to millions. Some of the observed negative tactics are listed below.

Misspelled paragraph or words

Changing message content

Increasing message volume

New delivery mechanism

Sentences without space

Inviting for the party

Usage of 'Dear friend' in the beginning of the mail

Subject line with upper case

Subject line ends with many exclamation marks

Replacing '.' with '-' in the email ID

The war is still continuing. As a result, many have done the research on statistical filters using Bayesian approach [8,15,21]. Interestingly, Gregory L. Wittel, and S. Felix Wu have come out with methodologies to beet statistical filters in their research paper[19]. Our interest is to develop statistical softwares, on similar lines to Paul Graham and Yerazunis, for huge corpus of mails.

E-mail spam is a subset of spam that involves sending nearly identical messages to numerous recipients by e-mail. [1,2,3,4,5]. However, initially the definition 'unsolicited commercial mail' was proposed in CAUCE [6]. Paul Jaudge descirbes a situation where in why companies are eager to make use of Spams as there daily routine hobby. According to him a company is selling wonky dolls for 50 dollars a doll. If the company lets the spammer send out 10 million mails and the response rate is just 0.1% it will make half a million dollars [7]. The rate of increase in the use of Spams by the companies is provided in references [8,9,10,11,12,13,14]. Spams were sent to 90 billion addresses per day in 2007, February.

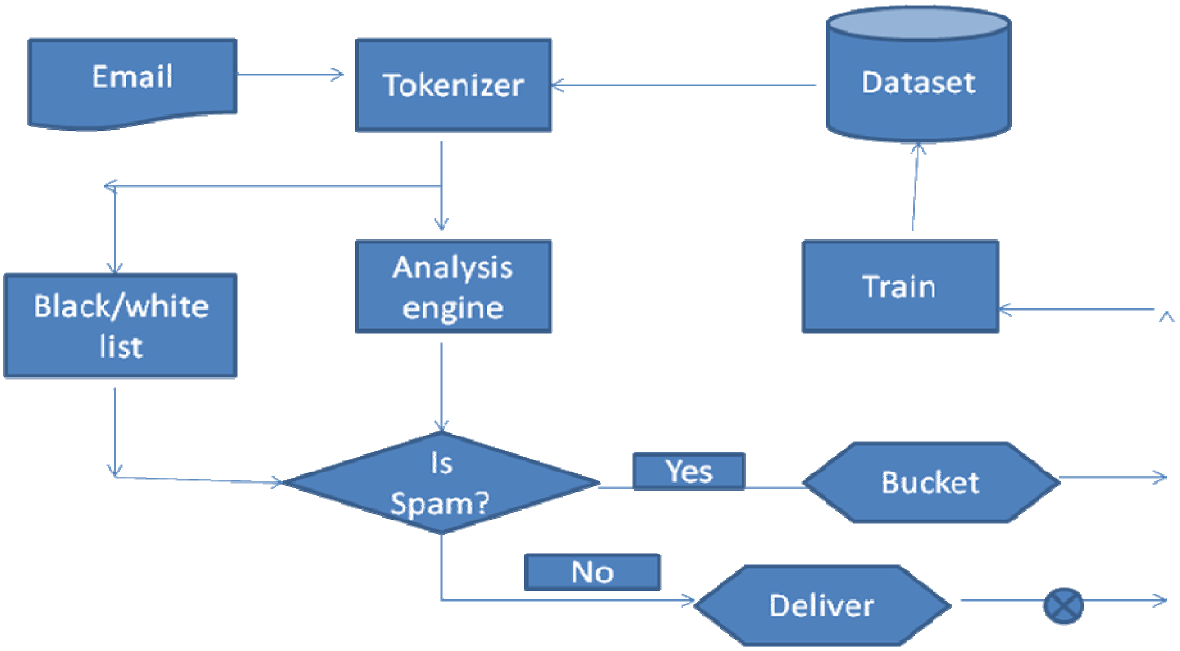
The use of filters is the main form of defense against spam. There are two groups of filters, Heuristic and Bayesian [15]: "The filters are called "heuristic" because they determine only the probability that a message may be junk e-mail, based on rules created from empirical observation of thousands of junk e-mail messages"[16]. The original heuristic spam filters such as Spam Assassin and Bright mail are weighted filters whose features are generated by humans working in a reactive way. These filters use a fixed set of feature detectors, and set a threshold score for rejection of an email as "too spammy". The first generation of these filters also had a human-generated weight set; now most heuristic filters use a neural net or other relaxation-based system to optimize the feature weighting [17].

A Bayesian filter needs no human intervention to generate the feature recognizers. By breaking the incoming text

into words, each word becomes a feature [18, 19]. The Bayesian filter counts the features (divided into spam and non-spam) frequencies and from that ratio determines a local probability on a message [20]. The latest range of filters decodes e-mail to its eye-space message before filtering. This use of eye-space is what gives filters a distinct advantage. Although spammers may do a lot to disguise their spam from filters, they cannot disguise their sales pitch without distorting it enough to render it ineffective [15]. Every other area of a message can be modified extensively; they do not have to adhere to the eye-space rule. Because the target is human, to capture this Bill Yerazunis originally introduced Sparse Binary Polynomial Hashing (SBPH), which is an approach to tokenization using word pairs and phrases.

DESIGN AND RESULTS

The schematic diagram of the process of a Spam filter is as follows.



First the email is passed through the tokenizer. Here email is busted into smaller components known as words/tokens. The tokenizer queries the dataset to identify the importance of each component and passes this information to the analysis engine or it passes through the black/white-list to take the decision immediately whether an email is spam or not. Black/white-list is a set of tokens where the end user customer is sure of tokens those lead to the decision Spam/Ham. The creation of the same is in the discretion of the end user. On the other hand, the analysis engine then calculates the disposition of the message into spam or non-spam. Based on the decision the additional action such as delivering or training a message can be taken.

Thus there are three central components to a filter.

1. Historical Dataset

It's a filter's memory. It contains a rather large catalog of characteristics that the filter has learned over a period of time to identify the most important characteristics (of spam/ham) of a user's email. The historical dataset of a statistical filter (actually the training process) contains counters to record the number of times each identifying mark was found in

each class of text, the number of messages analyzed and other information necessary to determine the nature of each characteristic. Suppose a classifier finds a new type of spam, it identifies new words and phrases, and these words or phrases are immediately updated into the dataset. For example, words like ‘Viagra’ and ‘call now’ discovered in spam are recorded. Over time, the recurrence of these words makes them excellent identifiers of spam. Innocent words are identified in the similar fashion. In the sense that words like ‘calculus’ or ‘Africa’ are stored in the dataset are identified as legitimate characteristics.

2. Tokenizer

The filter’s eyes. It is responsible off reading and interpreting an email, which it does by breaking it down into smaller components (tokens). The tokenizer works with the dataset to determine the significance of each token.

3. Analysis Engine

It is the filter’s technical part. Selects every token of an email and weighs them to determine whether an email is spam using the specified algorithm.

Here, the technical aspects of two types of filters are discussed, viz i. Bayesian filter and ii. MCMC filter.

In Bayesian filter, Paul Graham [18] approach is followed in order to generate probabilities associated with words. A Matlab program has been developed for testing purpose. To develop Spam filter using Bayesian approach, the process followed is mentioned below:

To start with, create SPAM and HAM corpus. A Corpus is a mass of E-mails. Corpus (wordlist, database, lexicon, or dictionary) needs to be developed based on the application. As soon as the corpus is created, then convert each Email into a workable format. In the sense that, all the E-mails are converted to text files by removing unwanted special characters, extra space between words etc... To cope up with the difficulties posed by numeric values (in matlab) appeared in the E-mail, convert the numerals into strings by prefixing numerals with the word ‘zero’. The text files (converted emails) are allowed to pass through a tokenizer (basically a command or a set of commands), which is responsible for reading and interpreting an email, which it does by breaking it down into smaller components (called tokens). A token can be a word, phrase, header, web address, or any other small piece of text in an email. The tokenizer works with the corpus dataset to determine the significance of each token. Later the Analyzer engine chooses key characteristics in an email and weighs them to determine whether or not a message is spam. Analyzer engine requires the token values to take the decision on the email. Token values represent the level of spamminess or in other words, the probability that a given token is an identifier of spam. To calculate the spamminess for each word or token (w) that appears in the corpus, the Paul Grahams formula is used:

$$p(w) = b(w)/(b(w)+g(w))$$

Where b(w) = (the number of spam emails containing the word w) / (the total number of spam email mails) and g(w) = (the number of ham emails containing the word w) / (the total number of ham email mails)

Tokens with a probability higher than a neutral 0.5 are considered spammy while tokens with a lower probability are considered hammy. Extreme cases are interesting i.e. tokens with a stronger disposition having values closer to 0 or 1.

It's unfair to take the decision on a new mail as spam or ham, just based on the values of p(w)= 1 or 0, due to lack of information on the tokens. Our own background information guides us here. As a result, it is followed that the Bayesian approach to determine an appropriate degree of belief about whether, when the word is seen again, it will be in a spam.

Robinson has given the formula to deal with rare words as follows:

Robinson_prob(w) = F(w) = (s*x+(n*p(w)))/(s+n)

where s and x are chosen to optimize performance. s is the strength we want to give to our background information, x being the assumed value of a token when n=0. A good starting value for s and x are 1 and 0.5 respectively. n being the number of emails has been received that contain word w. This yields a slight improvement in accuracy for many types of hard-to-classify messages, when a very little data has been collected for a particular token. Particularly useful in improving the identification of legitimate mail, thereby reducing false positives.

A static value is assigned to single-corpus tokens i.e. for tokens those appear in only one corpus. Assign a value p(w) = 0.99 to tokens appearing only in spam corpus and p(w)=0.01 to tokens appearing only in legitimate mail.

Missing innocent mails is worse than receiving SPAMS. Thus to prevent false positives Graham [18] has doubled the occurrence of tokens in legitimate messages.

There are tokens those are never seen before, or for which data are not collected. These tokens are 'hapaxes'. A hapaxial value (ranging between 0.4 or 0.5) will be assigned for hapaxes. It serves two purposes. First, it prevents unknown tokens from directing the outcome of the classification. Secondly, a threshold (minimum of five occurrences) is necessary to ensure honesty in our statistics and to avoid a situation in which the lack of data skews the overall results.

Now, tokenization process leaves us with several tokens with distinct values. In other words, each e-mail is represented by a set of probabilities. It is required to combine these individual probabilities into an overall combined indicator of spamminess or hamminess for the e-mail as a whole. The filter uses a sorting algorithm (descending order) to reorder the tokens so that the most interesting ones are toward the top, since these give us the best information about the subject email. Typically, 15 most interesting tokens are considered to build a decision matrix (some uses all the token values). To get a combined probability, it is looked at each token in our decision matrix as an independent test and use Graham's approach to combine individual probabilities to produce a single outcome i.e. to find the spammines of an email.

(F(w1)*F(w2)*…*F(w15)) / (F(w1)*F(w2)*…*F(w15))+((1-F(w1))*(1-F(w2))*…*(1-F(w15)))

Any value greater than or equal to 0.90 is an indicator of Spam else a legitimate mail.

Suppose a new mail comes, then tokenize and calculate F(w). Suppose a new token exists then a hapaxial value is assigned and calculate the probability of spamminnes of the email.

Existing, now-a-days, lots of rule based SPAM filters, however our interest is to develop statistical filter that uses Bayesian and MCMC approach. A Bayesian filter needs no human intervention to generate the feature recognizers. By breaking the incoming text into words, each word becomes a feature. The Bayesian filter counts the features (divided into spam and non-spam) frequencies and from that ratio determines a local probability on a message [1]. In order to test the technique, a matlab programme [3] has been written. The programme is divided into two parts. 1. Tokenizer and

2. Analyzer.

A set of red coloured codes constitute Tokenizer and the rest is Analyzer engine. The tokenizer reads email one by one and creates a dataset. Later, once again reads email one by one, tokenizes each mail and passes the information to the Analyzer engine. The main codes for the tokenizer are “total_feature=lower(textread(line,'%s'));” and “t=lower(textread(line,'%s'));”. The first code creates the dataset and the later tokenizes each mail. Textread(.,’%s’) function reads a white-space or delimiter-separated string from text file. For more information kindly go through Matlab manual.

```
clc
clear
fprintf('Enter no. of Spam mails as n=\n');
fprintf('Enter no. of mails as m=\n');
clear;
n=15;
m=65;
fid=fopen('out.txt','w')
fid1=fopen('file.txt','r')
    line=fgetl(fid1);
    total_feature=lower(textread(line,'%s'));
   fprintf(fid,'%s\n',total_feature{:});
    while ~feof(fid1)
        line=fgetl(fid1);
        total_feature=lower(textread(line,'%s'));
       fprintf(fid,'%s\n',total_feature{:});
end;
fid2=fopen('out.txt','r')
total_feature_out1=textscan(fid2,'%s');
total_feature_out=unique(total_feature_out1{:});
p=length(total_feature_out);
fid1=fopen('file.txt','r')
line=fgetl(fid1);
t=lower(textread(line,'%s'));
l=length(t);
```

For testing purpose, Corpora of mails consisting of 15 SPAMS and 50 HAMS were selected. Tokenizer in total generated 11305 features. All these features are saved in ‘out.txt’. To start with this collection of 11305 is considered as the dataset. A sample of the same is shown below.

genuie
and
from
the
original
manufacturer
are
selling
by
trusted
vendor
don't
miss

this
chance
to
buy
hight-quality
production
our

Analyzer compares the dataset with the information supplied by the tokenizer to find the spamminess of each word and finally calculates the spamminess of the entire email. For more details on the mathematical aspects kindly refer to ‘SPAM filter research report’.

```
mat_conv=zeros([l length(total_feature_out)]);
for k=1:length(total_feature_out);
for j=1:l;
mat_conv(j,k)=strcmp(total_feature_out{k},t{j});
end;
end;
mat_conv_trp=mat_conv';
occur_mat=sum(mat_conv_trp,2);
for i=1:length(occur_mat)
   if occur_mat(i) > 0
      occur_mat(i)==1;
   else occur_mat(i)==0;
   end;
end;


while ~feof(fid1)
   line=fgetl(fid1);
t=lower(textread(line,'%s'));
l=length(t);

clear mat_conv;
mat_conv=zeros([l length(total_feature_out)]);
for k=1:length(total_feature_out);
for j=1:l;
mat_conv(j,k)=strcmp(total_feature_out{k},t{j});
end;
end;
clear occur_mat1;
occur_mat1=zeros(l,1);
mat_conv_trp=mat_conv';
occur_mat1=sum(mat_conv_trp,2);
for i=1:length(occur_mat1)
   if occur_mat1(i) > 0;
      occur_mat1(i)==1;
   else occur_mat1(i)==0
   end;
end;
occur_mat=[occur_mat occur_mat1];
end;
occurmat=isfinite(1./occur_mat);
occurmat=[occurmat sum(occurmat(:,1:n),2) sum(occurmat(:,n+1:m),2)];
LocalProb_spam_word=(occurmat(:,m+1)./n)./((occurmat(:,m+1)./n)+2*(occurmat(:,m+2)./(m-n)));
```

```
%total_occurmat=sum(occurmat,2);
%robinson_prob=(0.5+(total_occurmat.*LocalProb_spam_word))/(1.+total_occurmat);
%occurmat=[occurmat robinson_prob];
occurmat=[occurmat LocalProb_spam_word];
occurmat=[occurmat occurmat(:,m+1)+2*occurmat(:,m+2)];
for i=1:length(occurmat(:,1));
    if((occurmat(i,m+1) +2*occurmat(i,m+2))< 5)& (occurmat(i,m+1)==0)
            occurmat(i,m+3)=0.45;
    else if((occurmat(i,m+1) +2*occurmat(i,m+2))< 5)& (2*occurmat(i,m+2)==0)
            occurmat(i,m+3)=0.45;
        else if  ((occurmat(i,m+1)+2*occurmat(i,m+2))>=5)& (occurmat(i,m+1)==0)
        occurmat(i,m+3)=0.01
            else if ((occurmat(i,m+1)+ 2*occurmat(i,m+2))>=5)& (2*occurmat(i,m+2)==0)
            occurmat(i,m+3)=0.99;
         end;
       end;
        end;
    end;
end;
fid0=fopen('Result65.txt','w');
 NSSC=0;
 NS=0;
   MS=zeros(p,1);
MH=zeros(p,1);
CS=zeros(p,1);
CH=zeros(p,1);
for j=1:m
    for i = (occurmat(:,j) ~= 0)
        occur_spam_prob=zeros([length(occurmat(i)) 2]);
        occur_spam_prob=[occurmat(i,j) occurmat(i,m+3)];
        occur_spam_prob=sort(occur_spam_prob, 'descend');
        occur_spam_prob=[occur_spam_prob 1-occur_spam_prob(:,2)];
    end;
    a=length(occur_spam_prob(:,1));

    if a>=10
        prob= (prod(occur_spam_prob(1:10,2)))/(prod(occur_spam_prob(1:10,2))+prod(occur_spam_prob(1:10,3)));
        fprintf(1, 'prob_spam_e%d=%f\n',j,prob);
        fprintf(fid0, 'prob_spam_e%d=%f\n',j,prob);
    else  prob= (prod(occur_spam_prob(1:a,2)))/(prod(occur_spam_prob(1:a,2))+prod(occur_spam_prob(1:a,3)));
        fprintf(1, 'prob_spam_e%d=%f\n',j,prob);
        fprintf(fid0, 'prob_spam_e%d=%f\n',j,prob);
    end;
    if prob < 0.9
        fprintf(1,'Email%d = %s\n',j,'HAM')
        fprintf(fid0,'Email%d = %s\n',j,'HAM')
        if j<=n
            MS=MS + occur_mat(:,j);

        else CH=CH + occur_mat(:,j);

        end;
    else if prob >= 0.9
            fprintf(1,'Email%d = %s\n',j,'SPAM')
            fprintf(fid0,'Email%d = %s\n',j,'SPAM')
            NS=NS+1;
            if j<=n
```

```
            NSSC=NSSC+1;
            CS=CS + occur_mat(:,j);
          else MH=MH + occur_mat(:,j);

         end;
      end;
   end;
end;

TS=(CS+MS);
i=(TS==0);
TS(i)=0.1;
Skew_factor_Spam=0.5+(MS./TS)./2;
TH=(CH+MH);
i=(TH==0);
TH(i)=0.1;
Skew_factor_Ham=0.5+(MH./TH)./2;
Prob_token_skew=(Skew_factor_Spam .* Skew_factor_Ham)./((Skew_factor_Spam .* Skew_factor_Ham)+((1-
Skew_factor_Spam).*(1-Skew_factor_Ham)));
```

Finally the filter gives as the misclassification table. The codes are as follows.

```
fprintf(1,'--------------Misclassification Table--------------\n');
fprintf(fid0,'--------------Misclassification Table--------------\n');
fprintf(1,'\t \t | SCC=%d | SMC=%d |\n \t \t -----------------\n \t \t | HMC=%d | HCC=%d |\n\n',NSSC,n-NSSC,NS-NSSC,m-
n-NS+NSSC);
fprintf(fid0,'\t \t | SCC=%d | SMC=%d |\n \t \t -----------------\n \t \t | HMC=%d | HCC=%d |\n\n',NSSC,n-NSSC,NS-
NSSC,m-n-NS+NSSC);

fprintf(1,'Overall Error Rate = %d\n',(n+NS-2*NSSC)/m)
fprintf(fid0,'Overall Error Rate = %d\n',(n+NS-2*NSSC)/m)
```

The misclassification table is as follows with an error rate.

```
--------------Misclassification Table--------------
        | SCC=15 | SMC=0 |
        -----------------
        | HMC=8 | HCC=42 |
```

Overall Error Rate = 1.230769e-001

Markovian classifier uses Sparse Binary Polynomial Hashing (SBPH) methodology with the weighting series of the order 2^{2n} instead of common weight of 1. The motto of SBPH is to create a lot of distinctive features from an incoming text, which actually break the incoming text into short phrases from one to five (a default window length) words each [15]. For a window of length N, this generates $2^{N}-1$ features. Each of these joint features can be mapped to one of the odd binary numbers from 1 to $2^{N}-1$ where original features at "1" positions are visible while original features at "0" positions are hidden and marked as skipped. Thus the SBPH generates a huge feature base. Now the question is that whether possible to use a smaller database thereby increasing speed and decrease the memory requirement and, of course, by maintaining the accuracy of the result. In order to address the issue mentioned above, I slightly altered the technique by considering only the combinations of adjacent features without place holders between the features, with an additional

advantage of reducing the feature set. The reason behind the change adopted in the technique is the assumption of single space between the features in the input text. The dataset can still be reduced using the calibration measure, and is calculated in the programme; however it is not used at the moment.

Thus the filter can be tightened to the extent as many phrases as we can make. In making phrases, the word order is more important than what words are used. These phrases, once tokenized, are inserted into the dataset in a fashion similar to that used in standard Bayesian analysis. Experimentally, it is found that the resulting accuracy increases as the weight increases based on the number of words in a phrase. Question arises on the extent of extra weight that the each sub phrase carries. Currently the best weight is the exponential super-increasing weights based on the weighting series 2^{2n} , used by Yerazunus [15]. Options are still open to come out with the better weights.

In the process of finding out the local probabilities, instead of counting matches of single words, matches of words and word phrases are counted , up to the predetermined window length (i.e. 5 words). Next, look up for matched words in the mails, convert the relative counts of matched pairs to local probabilities, then use a standard Bayesian algorithm to combine the local probabilities to find the spamminnes of a mail. The local probability formula is as follows:

$$P(w) = 0.5+(((Nspam – Nnonspam)*Weght)/((Nspam+Nnonspam)*WeightMax))$$

A token may have a probability of .95, but sometimes it doesn’t provide any information on the decision it is involved in previously. If a token has been involved in several erroneous classifications in the past, not only the token to be weighted differently than a reliable token, but also the token’s presence in a message to suggest that the results that generate may have a higher likelihood of being incorrect. Thus, Calibration algorithms provide additional weighting calibration to tokens in the dataset, based on a token’s reliability. Its ultimate goal is to avoid the misclassification in future by identifying the features that are the least effective at accurately indentifying spam, and simultaneously reduces the feature set accordingly. In other words, to find out how far in one particular direction (spam or ham) a token is likely to skew and is as follows:

$$K=(K^s *K^i) / ((K^s *K^i)+ (1-K^s)*(1-K^i))$$

Where K^s and K^i ‘s are skewing factor for both spam and legitimate emails respectively; a measurement of how far in one direction each token could possibly skew in the event of a misclassification.

$$K^s = 0.5+((M^s/(C^s + M^s))/2$$

And

$$K = 0.5+((M^i/(C^i + M^i))/2$$

M^s and M^i variables represent the total number of misclassifications in spam and legitimate mail, and C^s and C^i represent the total number of correct classifications in each corpus when the token was present in the message. Since .5 represents a neutral value, this probability shows no risk of skew. A value below 0.5 represent a skew in legitimate message and a value above 0.5 represents a skew in spams. The feature set is reduced based on the threshold set, usually threshold of (0.35, 0.65) is used. Any tokens which do not fall inside this interval can be eliminated from the feature set. A neutral probability of 0.5 can be assigned to a new token for which the sufficient information is not there.

Another methodology to reduce the feature set is to pass emails through a Black/white-list. A Black/white-listing is a common technique used to stop spam. Blacklist contains the addresses of spammers or spam words. When a message comes in, a check is performed to see if the sender/token is listed in the blacklist. If yes, the message is automatically treated labeled as spam. White-list is the opposite; they contain users that are verified contacts. These users may send messages that seem spammy, but because they are listed on the whitelist, will be treated as ham. Black/white-list is a technique that, employed by a large system such as MAPS, has been shown to catch only 24% of spam, with a very high false positive.

Following is the tokenizer of MCMC filter. It reads email one by one and tokenizes maintaining the order of the words appearing in the email using “textread(line,'%s')”. Later with a window of 5 words (default) at a time, it builds a feature set with the combination of the words. Suppose that first 5 words of a junk sentence is “one ear his posy hung “, then the tokenizer reads it and creates a feature set as ---one, one ear, one ear his , one ear his posy, one ear his posy hung .

```
clear;
clc;
fprintf('Enter no. of Spam mails as n=\n');
fprintf('Enter no. of mails as m=\n');
n=15;
m=65;
fid=fopen('fileout.txt','r')
fid1=fopen('file.txt','r')
fid3=fopen('total_out.txt','w')
while ~feof(fid1)
   line=fgetl(fid1);
   t=textread(line,'%s');
   l=length(t);
   line1=fgetl(fid);
   fid2=fopen(line1,'w')
   clear feature_phrase;
   feature_phrase=zeros(l,1);
   for i=0:(l/5)-1
      feature_chunk=t(5*i+1:5*i+5,1);
      xlswrite('feature_chunk',feature_chunk);
      [empty feature_chunk]=xlsread('feature_chunk.xls');
      for j=1:5
         if j==1
            feature_phrase=feature_chunk(j);
            feature_phrase{1}(length(feature_phrase{1})+1)=' ';
            fprintf(fid2,'%s\n',feature_phrase{:});
            fprintf(fid3,'%s\n',feature_phrase{:});
         else feature_phrase= strcat(feature_phrase,feature_chunk{j});
            feature_phrase{1}(length(feature_phrase{1})+1)=' ';
            fprintf(fid2,'%s\n',feature_phrase{:});
            fprintf(fid3,'%s\n',feature_phrase{:});
         end;
      end;
   end;
   if ~isempty(line),continue,end;
end;
```

For testing purpose, the same Corpora of emails used in the earlier case have been selected. A sample of the tokenizer result for the first email is as follows.

genuie
genuie and
genuie and from
genuie and from the
genuie and from the original
manufacturer
manufacturer are
manufacturer are selling
manufacturer are selling by
manufacturer are selling by trusted
vendor
vendor don't
vendor don't miss
vendor don't miss this
vendor don't miss this chance
to
to buy
to buy hight-quality
to buy hight-quality production
to buy hight-quality production our
online
online shop
online shop http://www.aurigaone.info
online shop http://www.aurigaone.info only
online shop http://www.aurigaone.info only today
and
and only
and only for
and only for you
and only for you all
prices
prices eased
prices eased for
prices eased for zero60
prices eased for zero60 p.s
forward
forward this
forward this email
forward this email to
forward this email to zero10
your
your friends
your friends and
your friends and you'll
your friends and you'll get
a
a good
a good bid
a good bid from
a good bid from us

Once the work for the entire email is done, then tokenizer passes the information to Analyzer engine. Analyzer

compares the features of each mail with the dataset and accordingly creates an occurrence matrix of every tokens/features and finally finds the number of spam mails and ham mails having the respective features. Later calculates the spamminnes of each token and hence the spamminnes of the entire email. Analyzer engine is as follows:

```
fid3=fopen('total_out.txt','r')
total_feature_out=importdata('total_out.txt');
p=length(total_feature_out);

fid=fopen('fileout.txt','r')
    line=fgetl(fid);
feature_phrase=importdata(line);
  u=length(feature_phrase);
    clear mat_conv;
    mat_conv=zeros([l p]);
    for k=1:p
        for j=1:u
            mat_conv(j,k)=strcmp(total_feature_out{k},feature_phrase{j});
        end;
    end;
    mat_conv_trp=mat_conv';
    occur_mat=sum(mat_conv_trp,2);
    while ~feof(fid)
         line=fgetl(fid);

 feature_phrase=importdata(line);
  u=length(feature_phrase);
        clear mat_conv;
        mat_conv=zeros([l p]);
        for k=1:p
            for j=1:u
                mat_conv(j,k)=strcmp(total_feature_out{k},feature_phrase{j});
            end;
        end;
        clear occur_mat1;
        occur_mat1=zeros(p,1);
        mat_conv_trp=mat_conv';
        occur_mat1=sum(mat_conv_trp,2);
        occur_mat=[occur_mat occur_mat1];
         if ~isempty(line),continue,  end;
end;
 occurmat=isfinite(1./occur_mat);
clear weight;
   weight=zeros(p,1);
 for i=1:p
weight(i,1)=2^(2.*(sum(isspace(total_feature_out{i})))-2);
   end;

occurmat=[occurmat sum(occurmat(:,1:n),2) sum(occurmat(:,n+1:m),2)];
LocalProb_spam_word=0.5+(((occurmat(:,m+1)- occurmat(:,m+2)).*weight)./((2*max(weight)).*(occurmat(:,m+1)+
occurmat(:,m+2)+1)));
%total_occurmat=sum(occurmat,2);
%robinson_prob=(0.5+(total_occurmat
%.*LocalProb_spam_word))/(1.+total_occurmat)
%occurmat=[occurmat robinson_prob];
occurmat=[occurmat LocalProb_spam_word];
fid0=fopen('ResultMCMC.txt','w');
```

```
 NSSC=0;
 NS=0;
    MS=zeros(p,1);
MH=zeros(p,1);
CS=zeros(p,1);
CH=zeros(p,1);
for j=1:m
    for i = (occurmat(:,j) ~= 0)
        occur_spam_prob=zeros([length(occurmat(i)) 2]);
        occur_spam_prob=[occurmat(i,j) occurmat(i,m+3)];
        occur_spam_prob=sort(occur_spam_prob, 'descend');
        occur_spam_prob=[occur_spam_prob 1-occur_spam_prob(:,2)];
    end;
    a=length(occur_spam_prob(:,1));

    if a>=10
         prob= (prod(occur_spam_prob(1:10,2)))/(prod(occur_spam_prob(1:10,2))+prod(occur_spam_prob(1:10,3)));
        fprintf(1, 'prob_spam_e%d=%f\n',j,prob);
   fprintf(fid0, 'prob_spam_e%d=%f\n',j,prob);
    else  prob= (prod(occur_spam_prob(1:a,2)))/(prod(occur_spam_prob(1:a,2))+prod(occur_spam_prob(1:a,3)));
        fprintf(1, 'prob_spam_e%d=%f\n',j,prob);
 fprintf(fid0, 'prob_spam_e%d=%f\n',j,prob);
    end;
    if prob < 0.9
            fprintf(1,'Email%d = %s\n',j,'HAM')
            fprintf(fid0,'Email%d = %s\n',j,'HAM')
      if j<=n
                MS=MS + occur_mat(:,j);

            else CH=CH + occur_mat(:,j);

            end;
    else if prob >= 0.9
        fprintf(1,'Email%d = %s\n',j,'SPAM')
        fprintf(fid0,'Email%d = %s\n',j,'SPAM')
        NS=NS+1;
        if j<=n
            NSSC=NSSC+1;
            CS=CS + occur_mat(:,j);

        else MH=MH + occur_mat(:,j);

        end;
    end;
    end;
end;

TS=(CS+MS);
i=(TS==0);
TS(i)=0.1;
Skew_factor_Spam=0.5+(MS./TS)./2;
TH=(CH+MH);
i=(TH==0);
TH(i)=0.1;
Skew_factor_Ham=0.5+(MH./TH)./2;
Prob_token_skew=(Skew_factor_Spam.*Skew_factor_Ham)./((Skew_factor_Spam.*Skew_factor_Ham)+((1-
Skew_factor_Spam).*(1-Skew_factor_Ham)));
```

The Analyzer engine outputs the result with the probability of spamminnes and categorizes to Spam if the probability is greater than or equal to .90 else to Ham. The result sheet is as follows.

prob_spam_e1=0.999983
Email1 = SPAM
prob_spam_e2=0.999960
Email2 = SPAM
prob_spam_e3=0.999983
Email3 = SPAM
prob_spam_e4=0.999498
Email4 = SPAM
prob_spam_e5=0.999983
Email5 = SPAM
prob_spam_e6=0.999983
Email6 = SPAM
prob_spam_e7=0.999983
Email7 = SPAM
prob_spam_e8=0.999908
Email8 = SPAM
prob_spam_e9=0.999983
Email9 = SPAM
prob_spam_e10=1.000000
Email10 = SPAM
prob_spam_e11=1.000000
Email11 = SPAM
prob_spam_e12=0.999960
Email12 = SPAM
prob_spam_e13=0.998830
Email13 = SPAM
prob_spam_e14=0.999983
Email14 = SPAM
prob_spam_e15=0.999983
Email15 = SPAM
prob_spam_e16=0.484380
Email16 = HAM
prob_spam_e17=0.498861
Email17 = HAM
prob_spam_e18=0.490778
Email18 = HAM
prob_spam_e19=0.489619
Email19 = HAM
prob_spam_e20=0.494466
Email20 = HAM
prob_spam_e21=0.437167
Email21 = HAM
prob_spam_e22=0.490236
Email22 = HAM
prob_spam_e23=0.498698
Email23 = HAM
prob_spam_e24=0.509960
Email24 = HAM
prob_spam_e25=0.498438
Email25 = HAM
prob_spam_e26=0.490778
Email26 = HAM

prob_spam_e27=0.490236
Email27 = HAM
prob_spam_e28=0.501823
Email28 = HAM
prob_spam_e29=0.491375
Email29 = HAM
prob_spam_e30=0.490778
Email30 = HAM
prob_spam_e31=0.489547
Email31 = HAM
prob_spam_e32=0.492080
Email32 = HAM
prob_spam_e33=0.491342
Email33 = HAM
prob_spam_e34=0.495508
Email34 = HAM
prob_spam_e35=0.491700
Email35 = HAM
prob_spam_e36=0.489585
Email36 = HAM
prob_spam_e37=0.491017
Email37 = HAM
prob_spam_e38=0.494792
Email38 = HAM
prob_spam_e39=0.492459
Email39 = HAM
prob_spam_e40=0.497396
Email40 = HAM
prob_spam_e41=0.491017
Email41 = HAM
prob_spam_e42=0.489422
Email42 = HAM
prob_spam_e43=0.492644
Email43 = HAM
prob_spam_e44=0.505273
Email44 = HAM
prob_spam_e45=0.496202
Email45 = HAM
prob_spam_e46=0.489374
Email46 = HAM
prob_spam_e47=0.486053
Email47 = HAM
prob_spam_e48=0.490063
Email48 = HAM
prob_spam_e49=0.495169
Email49 = HAM
prob_spam_e50=0.487551
Email50 = HAM
prob_spam_e51=0.492969
Email51 = HAM
prob_spam_e52=0.492291
Email52 = HAM
prob_spam_e53=0.490236
Email53 = HAM
prob_spam_e54=0.492080
Email54 = HAM
prob_spam_e55=0.498600

```
Email55 = HAM
prob_spam_e56=0.498698
Email56 = HAM
prob_spam_e57=0.489719
Email57 = HAM
prob_spam_e58=0.496528
Email58 = HAM
prob_spam_e59=0.498698
Email59 = HAM
prob_spam_e60=0.493924
Email60 = HAM
prob_spam_e61=0.492481
Email61 = HAM
prob_spam_e62=0.505273
Email62 = HAM
prob_spam_e63=0.496875
Email63 = HAM
prob_spam_e64=0.491342
Email64 = HAM
prob_spam_e65=0.492622
Email65 = HAM
```

Finally it calculates the misclassification table. The codes are as follows.

```
fprintf(1,'--------------Misclassification Table--------------\n');
fprintf(fid0,'--------------Misclassification Table--------------\n');
fprintf(1,'\t \t | SCC=%d | SMC=%d |\n \t \t -----------------\n \t \t | HMC=%d | HCC=%d |\n\n',NSSC,n-NSSC,NS-NSSC,m-n-NS+NSSC);
fprintf(fid0,'\t \t | SCC=%d | SMC=%d |\n \t \t -----------------\n \t \t | HMC=%d | HCC=%d |\n\n',NSSC,n-NSSC,NS-NSSC,m-n-NS+NSSC);

fprintf(1,'Overall Error Rate = %d\n',(n+NS-2*NSSC)/m)
fprintf(fid0,'Overall Error Rate = %d\n',(n+NS-2*NSSC)/m)
```

Below the misclassification table with overall error rate.

```
--------------Misclassification Table--------------
        | SCC=15 | SMC=0 |
        -----------------
        | HMC=0 | HCC=50 |
```

Overall Error Rate = 0

CONCLUSIONS

The Bayesian filter misclassifies 8 Ham mails which is a serious concern. In-fact the Bayesian filter works on the unigram model, unigram model decreases the accuracy of the result rather than increasing the same. However, this serious concern is taken care by MCMC filter which works on n-gram model. Though I have worked only on non-sparse sub feature phrases the result is satisfactory. Theory suggests using Orthogonal Sparse Bigram (OSB). The idea behind this approach is to gain speed by working only with an orthogonal feature set inside the window, rather than the prolific and probably redundant features generated by SBPH. It is to be noted that the redundant features can also be removed using calibration measures. Also, single feature do not provide sufficient information, hence by removing the same in the sliding

window feature set, the number of features (but not the same features, since the assumption of single space inputs) produced by the OSB and our technique is same. However, our result is satisfactory with the presence of unigram feature (single feature) in a sliding window feature set, and hence decided to go with it. Also, speed is reduced to almost 31%, since our technique produces exactly N (=5) features in a sliding window of length N, while SBPH generates 2^{N-1} (=16) sub phrases.

REFERENCES

1. James John Farmer (27 December 2003). 3.4 Specific Types of Spam (FAQ). *An FAQ for news.admin.net-abuse.email; Part 3: Understanding NANAE*. spamfaq.net. Retrieved on 2007-01-05.
2. You Might Be An Anti-Spam Kook If.... Rhyolite Software, LLC (25 November 2006). Retrieved on 2007-01-05.
3. On what type of email should I (not) use SpamCop? (FAQ). *SpamCop FAQ*. IronPort Systems, Inc... Retrieved on 2007-01-05.
4. Scott Hazen Mueller. What is spam?. *Information about spam*. Spam.abuse.net. Retrieved on 2007-01-05.
5. Spam Defined. Infinite Monkeys & Co. LLC (22 December 2002). Retrieved on 2007-01-05.
6. CAUCE, *How do you define spam?,* http://www.cauce.org/about/faq.shtml#how
7. Paul Judge, *The State of the Spam Problem,* Educase review 2003.
8. *a b c* IronPort Systems, Inc. (28 June 2006). *Spammers Continue Innovation: IronPort Study Shows Image-based Spam, Hit & Run, and Increased Volumes Latest Threat to Your Inbox*. Press release. Retrieved on 2007-01-05.
9. Brad Templeton (Tue, 08 Mar 2005 08:30:08 GMT). Reaction to the DEC Spam of 1978. Brad Templeton. Retrieved on 2007-01-21.
10. J.S. Kelly (1 September 2002). A brief history of spam. *developerWorks*. IBM. Retrieved on 2007-01-05.
11. David Streitfeld. "Opening Pandora's In-Box", Los Angeles Times (latimes.com), 11 May 2003. Retrieved on 2007-01-06. *free registration required to view article contents*
12. Staff. "Bill Gates 'most spammed person'", BBC News (bbc.co.uk), 18 November 2004. Retrieved on 2007-01-06.
13. Mike Wendland. "Ballmer checks out my spam problem", ACME Laboratories republication of article appearing in Detroit Free Press, 2 December 2004. Retrieved on 2007-01-06. *the date provided is for the original article; the date of revision for the republication is 8 June 2005; verification that content of the republication is the same as the original article is pending*
14. Jef Poskanzer (15 May 2006). Mail Filtering. ACME Laboratories. Retrieved on 2007-01-06.
15. William S. Yerazunis, PhD , The Spam-Filtering Accuracy Plateau at 99.9% Accuracy and How to Get Past It, 2004.
16. *Heuristic filters.* http://www.madgoat.com/mx/mx_mgmt_guide.html#heading_8.7

17. SpamAssassin , http://spamassassin.apache.org/

18. Paul Graham, *A plan for spam*, August 2002.

19. Paul Graham, *Better Bayesian filtering*, January 2003

20. Michael Bevilacqua-Linn, *Machine Learning for Naive Bayesian Spam Filter Tokenization*, 2003, http://www.cs.rochester.edu/u/brown/Crypto/studprojs/Spam.pdf

21. Gregory L. Wittel and S. Felix Wu. On Attacking Statistical Spam Filters. Department of Computer Science, University of California, Davis, CA 95616 USA.